Análisis de las prestaciones del Sistema Operativo Osekpara el Procesamiento de Señales y la Enseñanza de RTOS

Osio Jorge Rafael[1]; Salvatore Juan[1]; Mendez Leandro[2]; Morales Daniel Martín[1]
1. *Universidad Nacional Arturo Jauretche, Instituto de Ingeniería,*
Florencio Varela, Av. Calchaquí 6200
2. *Universidad Nacional de La Plata – Facultad de Ingeniería*
La Plata, Calle 1 y 47

Abstract

En este trabajo se realiza un estudio de las prestaciones del sistema operativo osek para la enseñanza de sistemas operativos de tiempo real y la utilización del mismo en aplicaciones industriales. En primera instancia se analizan las prestaciones del mismo que lo hacen propicio para la enseñanza de sistemas operativos de tiempo real y luego se mostrarán las bondades del mismo mediante la implementación de un sistema adquisidor de señales para el procesamiento y transmisión de las mismas. En base a la implementación se obtuvieron excelentes resultados e importantes consideraciones a tener en cuenta al momento de utilizar osek.

Palabras Clave: *Sistema Operativos de Tiempo Real, Sistemas Embebidos, Programación de alto nivel, Aplicaciones Industriales.*

1. Introducción

Este artículo tiene como objetivos analizar las prestaciones del Sistema Operativo OSEKy mostrar las prestaciones del mismo para una aplicación de procesamiento en tiempo real.

En una primera parte se hará una descripción detallada de las características del sistema operativo OSEK [1], en donde se desarrollarán las características principales, las bondades que ofrece y las limitaciones para determinados aspectos de tiempo real.

En base a esta descripción se presentarán las características didácticas que hacen que este sistema operativosea ideal para la enseñanza y la implementación de las características principales de todo RTOS (sistema operativo de tiempo real), como lo son la creación de tareas, procesos y la temporización dentro de un sistema operativo de tiempo real.

En la segunda parte del trabajo se describe la implementación de un sistema de adquisición y procesamiento de señales mediante el uso de la placa EDU_CIAA_NXP [2], para una aplicación de filtrado digital de audio (ver Fig. 1), empleando los periféricos ADC y DAC embebidos en el microcontroladorLPC4337 y un sistema operativo de tiempo real cuya implementación se basa en el estándar OSEK.

Entre las tareas a desarrollar sobre la implementación del sistemase tiene la configuración del sistema operativo a fin de implementar procesos que permitan la utilización de los pulsadores incorporados en la placa, el puerto DAC y uno de los canales ADC disponibles. Se describirá una tarea inicial, la cual se encargará de inicializar los periféricos para que el sistema comience a funcionar, realice la selección de uno de los cuatro filtros digitales implementados, y finalmente active la tarea analógica AnalogicTask definida dentro del código. Dicha tarea tiene el propósito de almacenar muestras de una señal de voz en un arreglo de datos, para su posterior filtrado y reconstrucción digital.

En base a ésta implementación se obtuvieron importantes resultados que se desarrollarán en la última parte del trabajo.

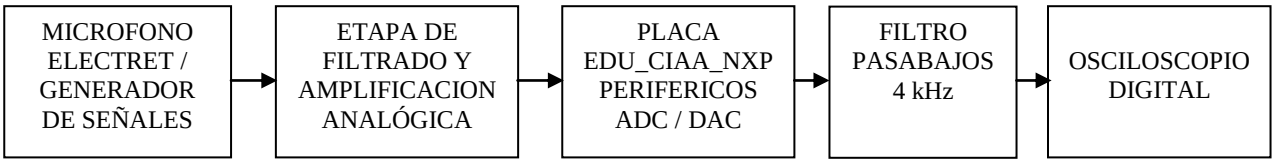

Figura 1. Diagrama en bloques de la aplicación

2. Características del sistema operativo Osek

Entre las principales ventajas de usar un RTOS se destaca la posibilidad de implementar un sistema multitarea, es decir que un hilo puede suspender su ejecución para que otro hilo se ejecute sin que el hilo original pierda su contexto. Existen aplicaciones en las cuales el tiempo de respuesta a un estímulo del sistema es un parámetro crítico, siendo requisito necesario que la respuesta del sistema se ubique dentro de una dada ventana de tiempo. Por ello el uso de un RTOS contribuye a aumentar el determinismo del sistema, además de incrementar su confiabilidad.

Para configurar el RTOS, en el estándar OSEK se definió otro estándar denominado OSEK ImplementationLanguage, comúnmente llamado OIL. Se trata de un lenguaje textual de muy fácil interpretación, similar al lenguaje C, donde se indican las características del sistema operativo, tales como las tareas que se van a ejecutar, las alarmas, interrupciones, etc.

En la Figura 2 se puede observar la configuración de una tarea denominada InitTask, la cual tiene una prioridad 1, 128 bytes de Stack, se auto inicia al comienzo, es de tipo básico y no se puede interrumpir durante su ejecución [1].

```
TASK InitTask {
PRIORITY = 1;
SCHEDULE = NON;
ACTIVATION = 1;
STACK = 128;
TYPE = BASIC;
AUTOSTART = TRUE {
APPMODE = ApplicationMode1;
     }
}
```

Figura 2. Configuración de una tarea

2.1. Tareas en Osek

En OSEK y por lo general en los sistemas de tiempo real las tareas realizan su cometido y terminan.

No se utilizan estructuras como while(1) para mantener la tarea en ejecución, ni sleeps para detener el microcontrolador entre activaciones. Para los casos en los que una determinada tarea requiera de un tiempo de espera, el OSEK provee de un mecanismo de eventos, mediante el cual la tarea entra en estado de suspensión a la espera de que se produzca el evento que necesita para finalizar su ejecución, permitiendo la ejecución de otras tareas. Para el control de las tareas, se utilizan las siguientes interfaces:

- ActivateTask: Permite activar una tarea
- TerminateTask: Guarda el contexto y finaliza una tarea
- ChainTask: Realiza la combinación de las dos anteriores

El sistema OSEK permite 2 tipos de tareas, BASIC y EXTENDED. Las tareas básicas no tienen eventos y por lo tanto no pueden permanecer en el estado de espera. A diferencia de estas, las tareas extendidas pueden tener uno o más eventos y pueden entrar en el estado de espera.

La prioridad de una tarea es un número entero entre 1 y 255 definido en el archivo OIL. Mientras mayor sea el número, mayor es la prioridad de ejecución. Si dos tareas tienen la misma prioridad, son ejecutadas según su orden de llegada al buffer FIFO. Una tarea que se encuentra corriendo nunca va a ser interrumpida por una de menor prioridad ni por una de la misma prioridad.

Existen dos configuraciones posibles para la programación de la ejecución de las tareas, la primera denominada NON PREEMPTIVE, en las cual no pueden ser interrumpidas nunca por otra tarea, sin importar que prioridad tengan. Las tareas NON PREEMPTIVE se ejecutan sin interrupción hasta que terminan, pasando luego al estado de espera. La segunda configuración, denominada PREEMPTIVE, permite que una tarea en ejecución sea interrumpida por otra de mayor prioridad, poniendo la tarea en estado de espera.

2.1.1. Estados de las Tareas en Osek. En el sistema operativo OSEK una tarea puede estar en cualquiera de los siguientes estados:

- En ejecución: La tarea se encuentra corriendo, utilizando los recursos del procesador en ese momento. En cada momento solo una tarea puede encontrarse en este estado.
- Lista: En este estado están todas las tareas que se encuentran esperando que se liberen los recursos del procesador para poder ejecutarse. No se encuentran en el estado de ejecución porque una tarea de mayor prioridad se encuentra utilizando el procesador.
- En espera: La tarea se encontraba corriendo y decidió esperar que ocurra un evento. Hasta que el evento que espera ocurra, la misma se encontrará en este estado.
- Suspendida: Es el estado por defecto de las tareas, es decir que la tarea esta momentáneamente desactivada.

2.2. Alarmas

El OSEK utiliza alarmas para realizar una acción luego de cumplirse un determinado tiempo. Las acciones pueden ser activar una tarea, setear un evento o realizar una retrollamada en C. Para la implementación de las alarmas, ya sea, que se utilizarán una o más alarmas, el sistema operativo utilizará un contador de hardware. Cada alarma debe ser definida en el archivo OIL así como su correspondiente acción. Para manejar las alarmas se utilizan las siguientes interfaces:

- SetRelAlarm: Establece una alarma al tiempo relativo actual.
- SetAbsAlarm: Establece una alarma en tiempo absoluto.
- CancelAlarm: Cancela la activación de una alarma.

En la Figura 3 se muestra el uso de dos alarmas activadas por la tarea TaskB. La alarma ActivateTaskCes configurada para expirar por primera vez luego de 100 ticks (cuentas)y luego de forma repetitiva cada 100 ticks, dicha alarma se encarga de activar TaskC. La alarma SetEvent1TaskA es activada para setear un evento en TaskA y expirar luego de 150 ticks la primera vez y luego cada 200 ticks.

```
TASK(TaskB)
{
/* código */
SetRelAlarm(ActivateTaskC, 100, 100);
SetRelAlarm(SetEvent1TaskA, 150, 200);
TerminateTask();
}
```

Figura 3.Definición de Alarmas

2.3. Interrupciones

El sistema operativo OSEK utiliza dos tipos de interrupciones:

- ISR1: (InterruptServiceRoutine) Se trata de las interrupciones de categoría 1, las cuales son transparentes al OSEK y por ello no pueden utilizar casi ninguna interfaz del sistema operativo.
- ISR2: Se trata de las interrupciones de categoría 2, las cuales tienen una mínima intervención del OS y por ello pueden utilizar algunas interfaces del sistema operativo.

Cualquier ISR ya sea de categoría 1 o 2, va a interrumpir cualquier tarea independientemente de la prioridad de la misma. La planificación de la ejecución de las tareas es realizada por el kernel del sistema operativo, mientras que las de las ISR son administradas por el hardware del microprocesador. Sin importar si se trata de una tarea de tipo PREEMPTIVE o NON PREEMPTIVE, la misma va a ser interrumpida si se recibe una interrupción. No está permitido llamar a ninguna interfaz del sistema operativo mientras las interrupciones están deshabilitadas, salvo estas funciones para habilitar y deshabilitar las interrupciones. En caso de querer evitar esto, el sistema operativo provee al usuario las siguientes interfaces para manipular las interrupciones:

- DisableAllInterrupts
- EnableAllInterrupts
- SuspendAllInterrupts
- ResumeAllInterrupts
- SuspendOSInterrupts
- ResumeOSInterrupts

3. Aspectos destacables para la enseñanza de RTOS

A diferencia de otros sistemas operativos como Linux y Windows, el OSEK es un sistema operativo estático, es

decir que las tareas, sus prioridades, la cantidad de memoria que utilizan y sus recursos son definidos antes de compilar el código, durante un proceso que se denomina generación. Esto hace que no sea posible crear una tarea de forma dinámica, ni cambiar la prioridad a una tarea durante el tiempo de ejecución. La finalidad de ello es realizar una tarea específica en tiempo y forma, donde la tarea a realizar está definida con anterioridad y no es necesario cargar nuevas tareas de forma dinámica. Estas características hacen que el Osek sea ideal para la enseñanza de sistemas operativos de tiempo real, debido a que se puede estudiar y realizar la ejecución de un sistema operativo dejando de lado la complejidad que incorpora el hecho de la asignación de memoria y la creación de tareas de manera dinámica.

En otros sistemas operativos las tareas se ejecutan durante un tiempo indeterminado hasta que son finalizadas, en OSEK y por lo general en los sistemas de tiempo real [3], las tareas realizan su cometido y terminan, esto simplifica mucho el aprendizaje de la creación de tareas y sus características. Cuando una tarea requiera de un suceso para continuar ejecutándose, dispondrá deun mecanismo de eventos, mediante el cual la tarea entra en estado de suspensión a la espera de que se produzca el evento que necesita para finalizar su ejecución, permitiendo la ejecución de otras tareas. Esto último es indispensable para comprender como es que una tarea libera el uso de los recursos del procesador cuando no está haciendo nada o está a la espera de un suceso.

Por último, este sistema operativo permite la implementación de alarmas de manera sencilla y a su vez provee interrupciones. En su conjunto, el Osek provee los elementos principales para la comprensión del funcionamiento y configuración de un sistema operativo de tiempo real, que a su vez sirve como base para introducir al alumno en el estudio de los Sistemas Operativos.

4. Implementación del sistema

El diseño del sistema consiste en configurar el sistema operativo freeOSEK, a fin de implementar tareas que permitan la utilización de los periféricos que se muestran en la Figura 1. La tarea inicial, se encarga de iniciar los periféricos a utilizar (DAC, ADC) y del ajuste de parámetros tales como la frecuencia de muestreo del conversor, timer del periférico DMA, uso de filtros digitales, y finalmente activar la tarea analógica AnalogicTask.

4.1. Implementación de filtros digitales

Con el objeto de eliminar el ruido que trae incorporada la señal de voz digitalizada se implementaron los filtros de *Butterworth*. Estos filtros fueron configurados por software en el Microcontrolador, luego de la etapa de digitalización y almacenamiento en memoria. Para la implementación de dichos filtros, se generaron los coeficientes correspondientes medianteel software MatLab [6]. Entonces, el filtro de *Butterworth*implementado tiene los siguientes coeficientes

1. Pasa bajos Butterworth de segundo orden, frecuencia de corte fc 1000Hz:

[a, b] = butter(2, 1000 / 4000);

a = {0.09763107, 0.19526214, 0.09763107}
b = {1.000000, -0.94280904, 0.33333333}

2. Pasa alto Butterworth de segundo orden, frecuencia de corte fc 2000Hz:

[a, b] = butter(2, 2000 / 4000, 'high');

a = {0.29289321, -0.58578643, 0.29289321}
b = {1.00000000, -0.00000000, 0.17157287}

4.2. Configuración de Hardware

Como etapa de adquisición de la señal de voz se utilizó un micrófono tipo electret, un filtro de Butterworth de segundo orden y una etapa preamplificadora con ganancia 10 basada en un amplificador operacional. La señal de salida se contiene una componente de continua de aproximadamente 1,6V con la finalidad de aprovechar todo el rango dinámico del conversor, el cual va desde 0 a 3,3V.

Parte de la configuración de los periféricos, se efectúa durante las rutinas de inicialización implementadas en el Firmware del proyecto CIAA.

Para el periférico ADC (conversor analógico digital) se invoca a la función Chip_ADC_Init de la capa driver, y su configuración resultante por defecto, establece:

- ADC_MAX_SAMPLE_RATE en 400kHz (frecuencia de muestreo)
- ADC_10BITS (tamaño de dato digitalizado de 10 bits)
- Modo burst deshabilitado (conversión continua)

No obstante, fue necesario efectuar ciertos cambios a fin de satisfacer los requerimientos de la aplicación.

Estos es, una frecuencia de muestreo de dos veces la frecuencia de señal para cumplir con el teorema del muestreo.

Mediante llamadas a la función ciaaPOSIX_ioctl del estándar [4], se configura:

- Tasa de muestreo de 8kHz con su parámetro ajustado en 50 ticks de reloj.
- Resolución del conversor a 10 bits.
- Habilitar el canal 1 del conversor.

Para la configuración del ADC [5], el parámetro pasado a la función ioctl, toma el siguiente valor:

Nº ticks = 400 kHz / 8 kHz = 50

Mediante llamadas a la función ciaaPOSIX_ioctl se realiza la configuración del conversor digital a analógico DAC, la cual tiene las siguientes características:

- Tasa de muestreo de 8kHz con su parámetro frecuencia ajustado en 8000Hz.
- Habilitar el canal 0 del conversor DAC.

Por último, se establece el temporizador del módulo DMA en 125 µs por muestra.

4.3. Configuración de Software

Es importante destacar que para la configuración personalizada del sistema operativo se deben conocer las características de las bibliotecas POSIX [4]. A continuación se muestra la configuración del Archivo OIL (OSEK ImplementantionLanguage).

```
OSEK OSEK {
OS       ExampleOS {
STATUS = EXTENDED;
ERRORHOOK = TRUE;
PRETASKHOOK = FALSE;
POSTTASKHOOK = FALSE;
STARTUPHOOK = FALSE;
SHUTDOWNHOOK = FALSE;
USERESSCHEDULER = FALSE;
MEMMAP = FALSE;
};

TASK InitTask {
PRIORITY = 1;
ACTIVATION = 1;
AUTOSTART = TRUE {
APPMODE = AppMode1;
}
STACK = 512;
TYPE = BASIC;
SCHEDULE = NON;
RESOURCE = POSIXR;
}

TASK Analogic {
PRIORITY = 5;
ACTIVATION = 1;
STACK = 2048;
TYPE = EXTENDED;
SCHEDULE = FULL;
RESOURCE = POSIXR;
EVENT = POSIXE;}
```

Además, en cuanto a cambios en la configuración del sistema operativo se efectuó una ampliación del tamaño de pila en la tarea AnalogicTask de STACK = 2048 con prioridad cinco.

Dado que la unidad mínima de tiempo disponible tiene el valor de 1ms, tanto parainterrupciones de systick como para las alarmas y temporizadores, con la implicancia que el control del programa pasa al planificador, se apagan las interrupciones y el usuario pierde el control sobre los periféricos utilizados durante al menos 1ms hasta que la tarea vuelva a planificarse.En consecuencia, para ajustar la planificación de la tarea analógica al requerimiento de esta aplicación, resultó necesaria la eliminación de la alarma AnalogicAlarm y solidariamente se invoca dicha tarea al final de InitTask, a fin de que el control del programa entre en un lazo infinito con las líneas de código read, filter y write.

```
TASK(Analogic) {
int32countrx;
uint16hr_ciaaDac[16];
/* Almacenamos 16 muestrasde 10 bits */
uint16salida[16];
uint8i;
filterType*but           =           &
(filterCfg.butter[filterCfg.ind] );
while(1)
{
      /* Read ADC. */
countrx    =    ciaaPOSIX_read(fd_adc,
&hr_ciaaDac, sizeof(hr_ciaaDac));
if(countrx> 0 )
{
for(i = 0; i<=countrx; i = i + 1)
{
filter(but,&hr_ciaaDac[i],&salida[i]);
}
             /* Write DAC */
ciaaPOSIX_write(fd_dac,        &salida,
countrx);
}
}
/* end of Analogic */
```

Figura 4. Tarea para reconstrucción de la señal

4.3.1. Modificaciones en ciaaDriverAio.c

Modificaciones en ciaaDriverAio_ioctl:

A fin de configurar la frecuencia de trabajo de los módulos conversores ADC y DAC [5], se efectuaron los cambios indicados en la Figura 4 dentrodel archivo fuente, para la obtención del parámetro “value”.

Modificaciones en void ciaa_lpc4337_aio_init(void):

Durante la inicialización se obtiene un canal libre para las transferencias DMA, el cual será utilizado durante todo el tiempo de ejecución del programa,es decir, que no se utiliza el multiplexado de canales.

*Modificaciones en staticvoidciaaDriverAio_dacIRQHandler(ciaaDevices_deviceTypeconst * constdevice):*

Se eliminó la llamada a la función Chip_GPDMA_Stop, la cual libera el canal y detiene las futuras transferencias. Dicha llamada debería incluirse en el cuerpo de la función device_close al momento de liberar el recurso.

5. Resultados

En base a los warnings del compilador se determinó que el parámetro 2 pasado a la función Chip_ADC_Int_SetChannelCmd era incorrecto, la para solucionar dicho inconveniente y seleccionar el canal correspondiente, se modificó el contenido en el miembro channel de la estructura ciaaDriverAioControlType del driver correspondiente al conversor ADC, esto solucionó un error de truncamiento en los datos, el cual fue difícil de detectar.

Respecto a las pruebas realizadas, la señal digitalizada fue procesada en la CIAA, donde se le aplicaron los filtros pasa-bajos y pasa-altos antes mencionados, con el objeto de eliminar el ruido.

Por último, se realizaron las simulaciones del filtro pasa bajos de segundo orden con frecuencia de corte 1kHz, BUTTER_2ND_LP_1000, donde la columna A es la secuencia de muestras original de una señal seno de frecuencia 100Hz, y la columna B es la secuencia filtrada como se muestra en la siguiente 5.

En la Figura 6 se muestran las simulaciones del filtro pasa bajos de segundo orden con frecuencia de corte 1kHz, BUTTER_2ND_LP_1000, donde la columna A es la secuencia original de una señal seno con una frecuencia de 1000Hz, (igual a la frecuencia de corte del filtro), y la columna B es la secuencia filtrada.

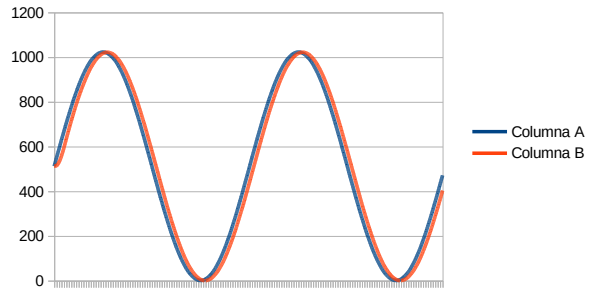

Figura 5. Simulación de la señal seno de 100Hz.

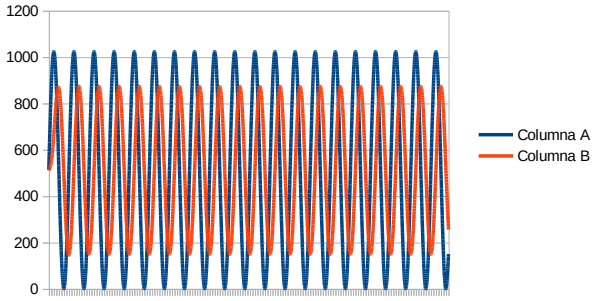

Figura 6. Simulación de la señal seno de 1000Hz

En base a estas simulaciones y a la posterior verificación de funcionamiento sobre la placa EDU-CIAA, se pudo comprobar el efecto del filtrado digital para distintas frecuencias que están dentro de los rangos de la señal de interés.

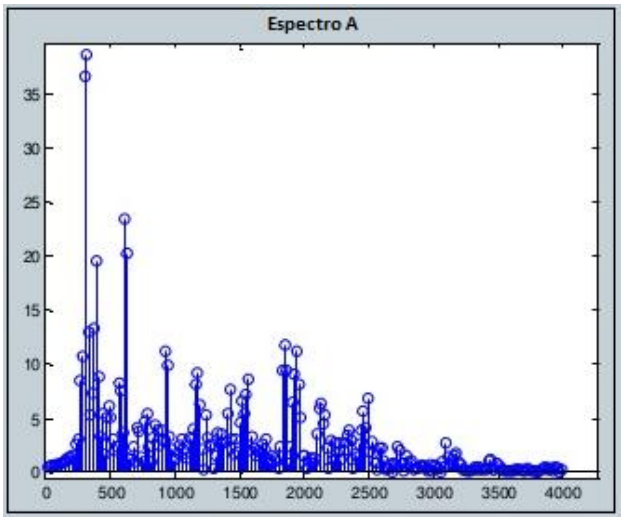

Figura 7. Espectro de una señal de audio digitalizada

Para comprobar el funcionamiento del sistema, los datos digitalizados de la señal de audio se transmitieron directamente mediante la uart de la CIAA a una

computadora. Estos datos fueron adquiridos usando Matlab y se graficó el espectro sin filtrar como se muestra en la figura 7. Para determinar el efecto del filtrado, se realizó el mismo procedimiento con la señal filtrada como se muestra en la figura 8. En dicha figura se puede observar como el efecto del filtrado eliminó las componentes de alta frecuencia.

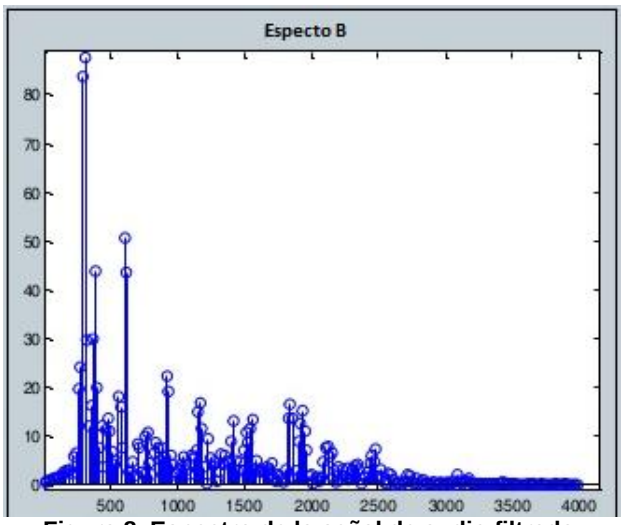

Figura 8. Espectro de la señal de audio filtrada

Habiendo analizado las señales de salida, se observó que la planificación de las tareas debía realizarse programando el manejo de los conversores DAC y ADC en el bucle infinito, a fin de cumplir con la tasa de muestreo. Considerando que la tarea analógica tiene un requerimiento de tiempo real de 125 µs, se determinó que la solución óptima se obtenía dejando al scheduler del osek la planificación de tareas con requerimientos de tiempo más extensos, tales como los fenómenos lentos de la detección de pulsadores para la selección de los filtros digitales.

6. Conclusiones

En primer lugar, se implementó un sistema de adquisición y procesamiento de datos que cumple con los requerimientos del problema planteado. Se comprobó mediante simulaciones y pruebas de adquisición con señales conocidas el correcto funcionamiento del sistema y la efectividad para filtrar el ruido.

Aunque la planificación y configuración del sistema operativo es muy intuitiva, es importante destacar que la implementación de los drivers de los periféricos no es flexible para aplicaciones personalizadas y es el único aspecto negativo del Sistema Operativo desde el punto de vista didáctico. En la Tabla 1 se muestra el nivel de complejidad que tiene el uso del sistema operativo OSEK en el diseño de una aplicación típica.

Tabla 1. Aspectos didácticos del sistema operativo.

Característica	Nivel de dificultad	Soporte
Tareas	bajo	alto
Alarmas	bajo	alto
Interrupciones	medio	medio
Drivers	alto	bajo

En cuanto a las limitaciones del osek, se determinó que el mismo no es óptimo para aplicaciones que requieren grandes retardos, como por ejemplo la implementación de protocolos de comunicación por software o la espera de un evento externo al sistema de manera indefinida. En aplicaciones con estas características, el OSEK puede dejar de funcionar. Adicionalmente, se determinó que las tareas que son demasiado rápidas deben implementarse fuera del scheduler para asegurar el cumplimiento de los tiempos.

Por último, se determinó que el Sistema Osek es ideal para la enseñanza de sistemas operativos, debido a las características básicas que posee y a la fácil configuración y ejecución por parte de los estudiantes. Esto último, se pudo verificar en un seminario dictado sobre el tema en el Instituto de Ingeniería de la carrera Ingeniería Informática de la UNAJ.

[1] Mariano Cerdeiro, “Introducción a OSEK-OS: El Sistema Operativo del CIAA-Firmware”, 2015

[2] ACSE, CADIEEL, “Computadora Industrial Abierta Argentina”, 2014.http://proyecto-ciaa.com.ar/

[3] Richard Barry, " Using the FreeRTOS Real Time Kernel", Renesas RX600 Edition, 2011

[4] Donald Lewine, "Posix Programmers guide",Orreilly, 1991

[5] “Sistemas Digitales”. R. Tocci, N. Widmer, G. Moss. Ed. Prentice Hall.

[6] Holly Moore, "Matlab para ingenieros", Editorial Mac pearson,2013